Computer Vision

Day 2

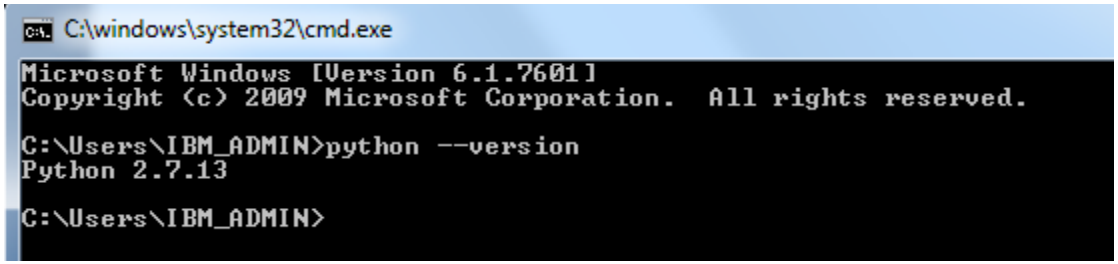
2017, Tanta University
Eng. Sara Hussien

Previous Session

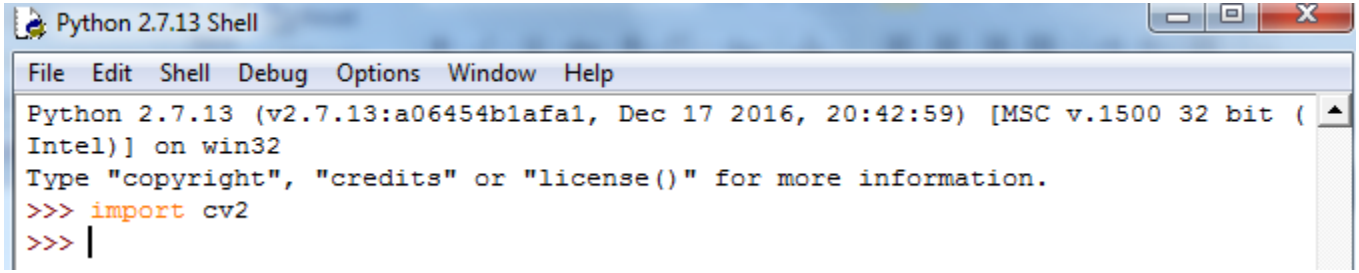
- Language Used : Python with OpenCV
- Tools -setup:
  1. Install python-2.7.13.msi
  2. numpy-v1.10
  3. matplotlib-1.3.0.win32-py2.7.exe
  4. opencv-3.2.0-vc14.exe ----> extract at c:/opencv
  5. after extracting Open CV copy file **cv2.pyd** from C:\opencv\build\python\2.7\x86 and past it to C:\Python27\Lib\site-packages
  6. class path setup for python
  7. start menu --> IDLE python --> check installation by write "import cv2"

Previous Session

- check for python installation



- check for OpenCV installation



Previous Session

- Read / write / display Image

➢ cv2.imread(filename) returns a uint8 image (values 0 to 255) unsigned 8 bit integer for pixel

Previous Session

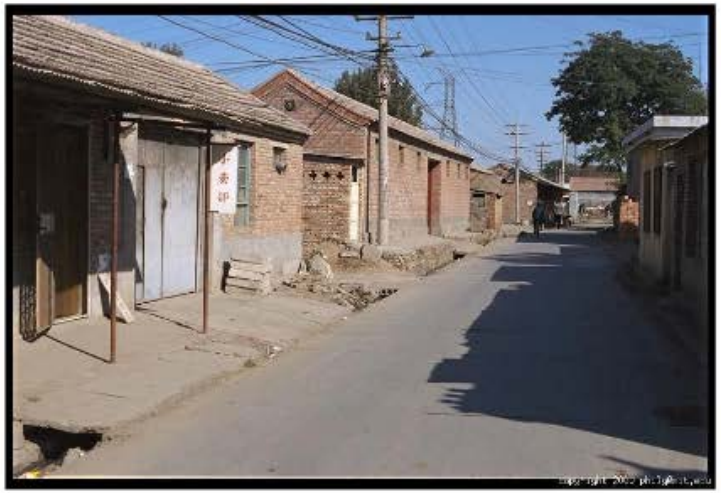
Image as a 2D sampling of signal

- Signal: function depending on some variable with physical meaning

- Image: sampling of that function
  - 2 variables: xy coordinates
  - 3 variables: xy + time (video)
  - 'Brightness' is the value of the function for visible light

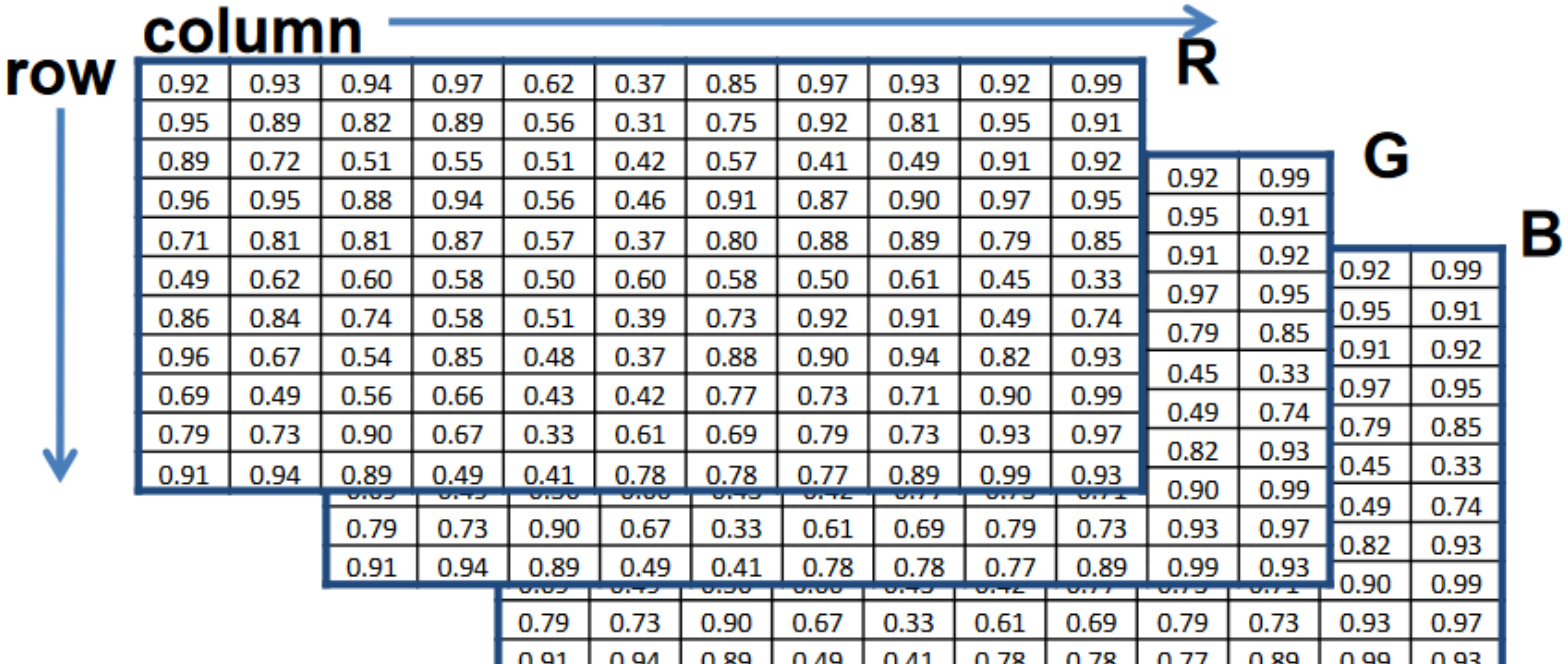
Previous Session

Color Image



Previous Session

- Images represented as a matrix
- Suppose we have a NxM RGB image called “im”:
  - im[1,1,0] = top-left pixel value in R-channel.
  - im[y, x, 1] = y pixels down, x pixels to right in the gth channel
  - Im[N, M, 2] = bottom-right pixel in B



Previous Session

- Some operations on Image:
  - Image.shape

```
height, width, channels = im.shape
print height, width, channels
```

  - Crop Image

```
#crop image
crop_img = im[100:300, 100:200]
```

  - Adding /subtracting two images

Previous Session

- Some operations on Image:
  - Display only one channel of colored image

```
#one channel of colored image
redimage=im[:,:,0]
greenimg= im[:,:,1]
blueimage=im[:,:,2]
```

  - Swap Colors pixels

```
#one channel of colored image
redimage=im[:,:,0].copy()
greenimg= im[:,:,1].copy()
blueimage=im[:,:,2].copy()

#swap red and blue pixels
im[:,:,0]=blueimage
im[:,:,2]=redimage
```

Image Filtering

- Image filters in spatial domain:
  - ✓ Filter is a mathematical operation of a grid of numbers
  - ✓ Smoothing, sharpening, measuring texture
- Image *linear* Filter :
  - ✓ Compute function of local neighborhood at each position

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

F[k,l] :is called the *kernel*, which is nothing more than the coefficients of the filter.

Example: box filter

Normalized Box Filter : This filter is the simplest of all! Each output pixel is the *mean* of its kernel neighbors.

$$f[\cdot,\cdot]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10							

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20						

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30					

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				
				?					

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30				
						?			
				50					

Image filtering

$f[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[\cdot,\cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Box Filter

What does it do?

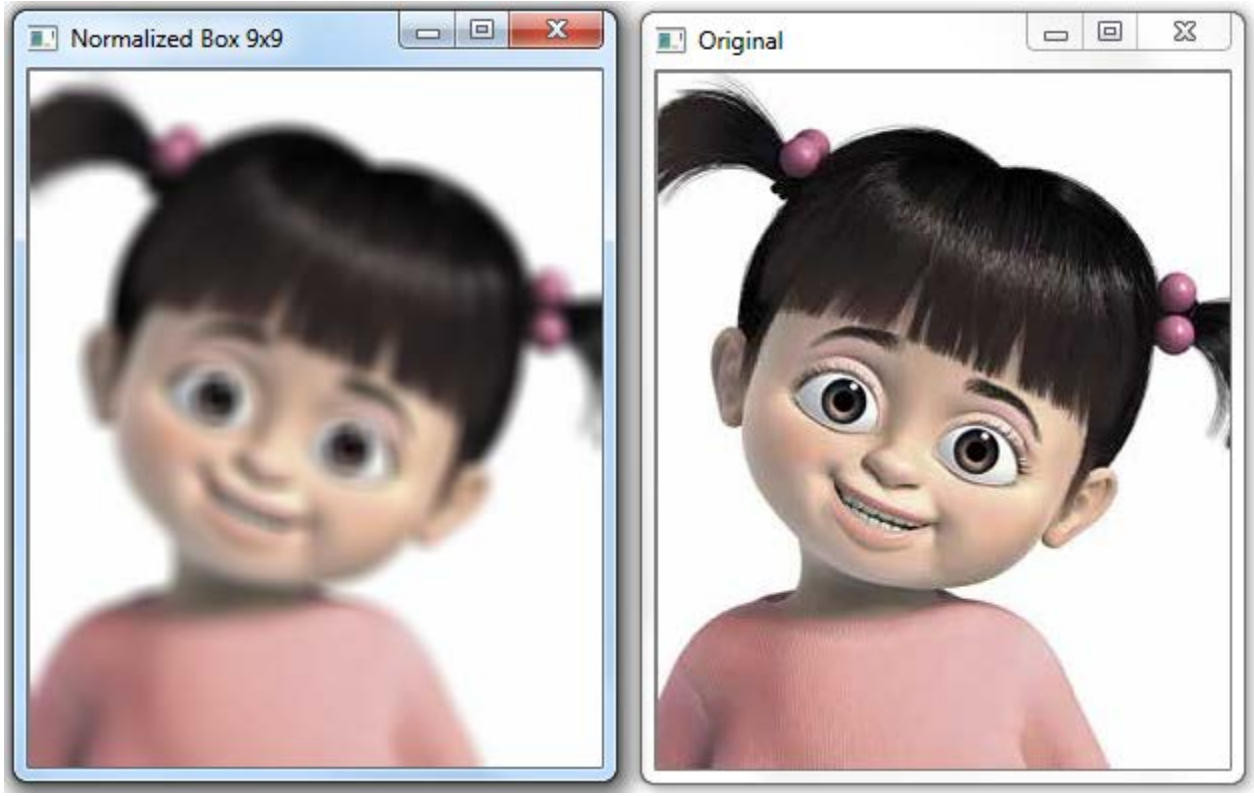
- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove sharp features)

$f[\cdot,\cdot]$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Smoothing with box filter



Think-Pair-Share time

1.

0	0	0
0	1	0
0	0	0

2.

0	0	0
0	0	1
0	0	0

3.

1	0	-1
2	0	-2
1	0	-1

4.

0	0	0
0	2	0
0	0	0

$-\frac{1}{9}$

1	1	1
1	1	1
1	1	1

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

2. Practice with linear filters

Original

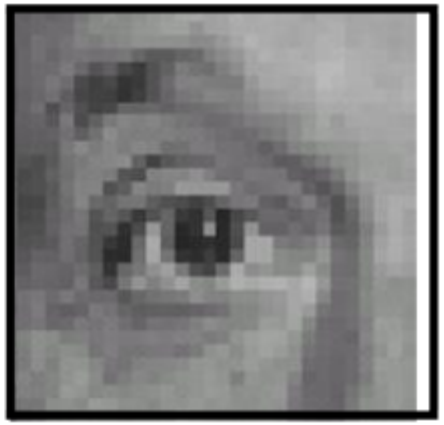
0	0	0
0	0	1
0	0	0

?

2. Practice with linear filters

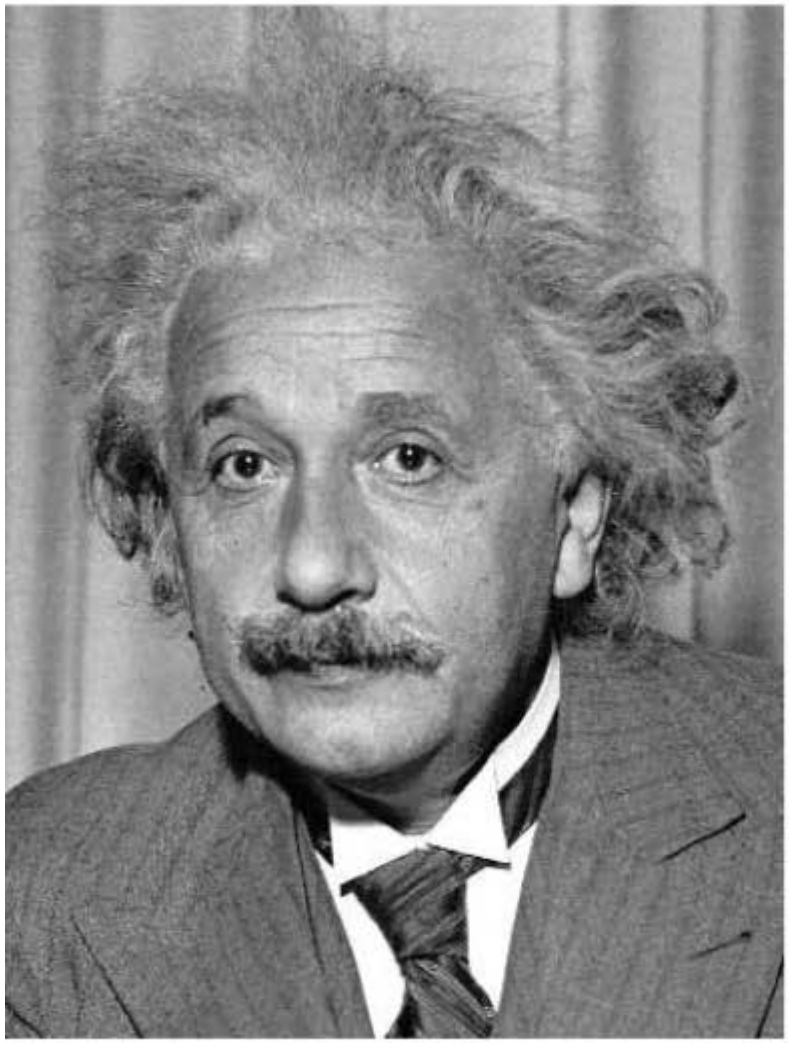
Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

3. Practice with linear filters

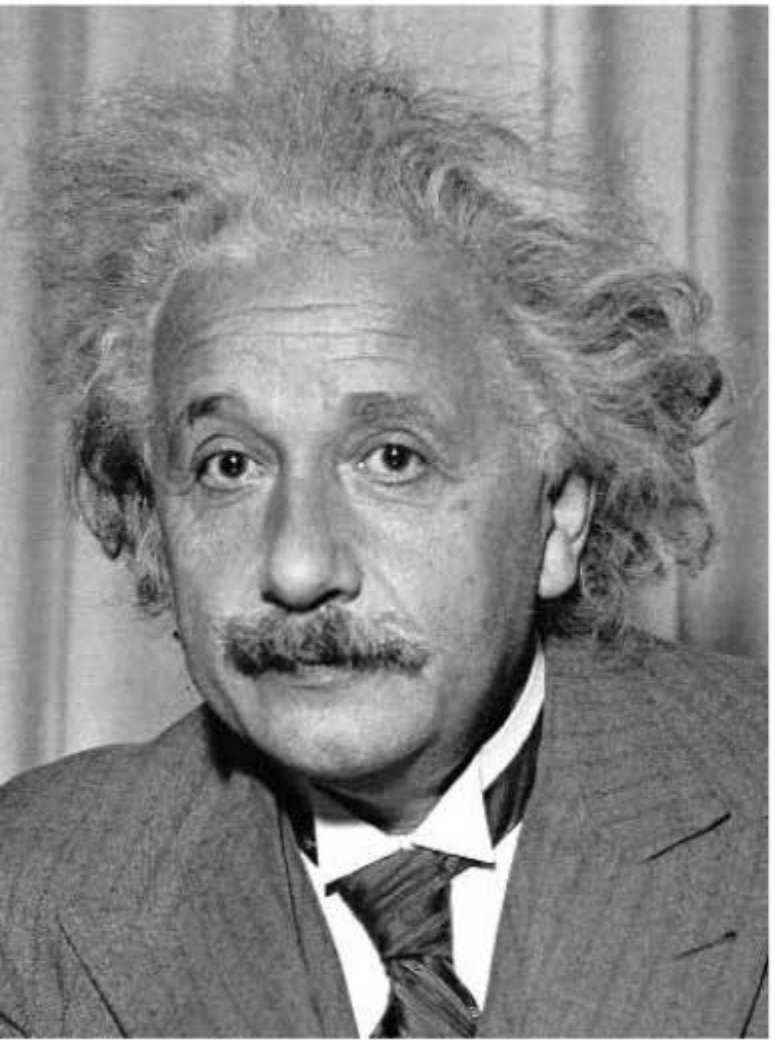


1	0	-1
2	0	-2
1	0	-1

Sobel

Vertical Edge
(absolute value)

3. Practice with linear filters



1	2	1
0	0	0
-1	-2	-1

Sobel

Horizontal Edge
(absolute value)

4. Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(Note that filter sums to 1)

?

4. Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

- Accentuates differences with local average

2D Convolution (Image Filtering)

- cv2.filter2D()

```
dst = cv2.filter2D(img,-1,kernel)
```

- **Image Blurring (Image Smoothing)**

1. **Averaging**

```
blur = cv2.blur(img,(5,5))
```

2. **Gaussian Blurring :**Gaussian blurring is highly effective in removing gaussian noise from the image.

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

3. **Median Blurring:** this is highly effective against salt-and-pepper noise in the images

```
median = cv2.medianBlur(img,5)
```